Vorlesungsmitschrift

Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

Inhaltsverzeichnis

1 Vorlesung 1 - 17.04.

1.1 Was sind Programmiersprachen?

Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.
Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

Memo technischer Begriff -> z.b. ADD reg1 reg2

Syntax einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)
- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

Semantik (diese Vorlesung)
- Bedeutung
- Interpretation
  Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

Fokus: Formalisierung

Pragmatik Fragen nach dem Gebrauch und Zweck (Useability).
Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)

1.2 Mehrdeutigkeit in natürlichen Sprachen

Synonyme *Schloss, Schimmel, ...*
Auflösung durch Kontext (meist leicht und unproblematisch)

Satzebene *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*
Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

Entwicklung der Formalisierungsansätze

operationale Semantik *(Landin 1964)*: Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnliche wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine

denotationelle Semantik *(McCarthy 1962)*: Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

axiomatische Semantik *(Hoare 1969)*: Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel: $\{Q\}P\{R\}$

1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)
Elementare Einheiten

```
// ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
Z::= 0 |1|...|MAX|-1|-2|...|MIN
// Wahrheitswerte BOOL
W::= TRUE | FALSE
// Konstanten KON
K::= Z | W
// Bezeichner bzw. Variablen mit Indizes
I::= a|b|...|z|a1|a2|...|zi
// Operatoren
OP::= + | - | * | / | mod
// boolesche Operatoren
BOP::= < | > | = | !> | !< | !=
```

Zusätzliche Einheiten *(induktiv)*

```
// Terme TERM
T::= Z  |  I  |  T1 OP T2 | read, für T1,T2 in TERM
// boolesche Terme BT
B::= W | T1 BOP T2 | read | not B
// Befehle (Zustandstransformation) COM
C::= skip | I := T | C1; C | if B then C1 else C2 | while B do C |
    output T | output B
```

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!
Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

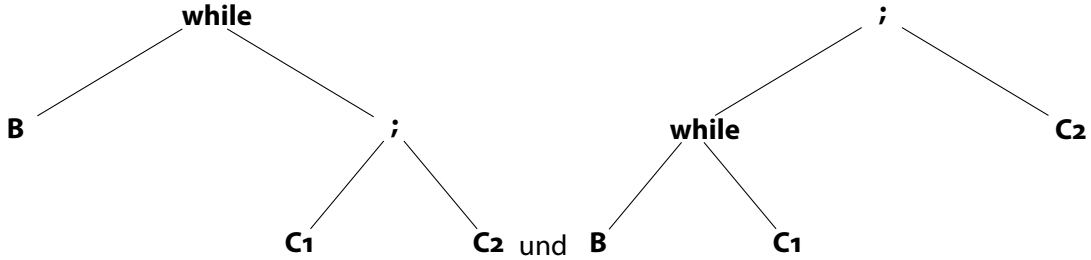
- Typkonflikte
- Fehlerbehandlung
- Rekursion

WHILE ist mehrdeutig?

Ja, das zeigt folgendes Beispiel:

```
while B do C1; C2
```

wo beide Syntaxbäume gültig sind.



Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

```
while B do
  C1;
  C2
```

2 Vorlesung 2 - 24.04.

2.1 weitor!